HOARE AXIOMS
AND THE SEMANTICS OF CONTRCL STRUCTURES

CS-1977-10

Edmund Melson Clarke, Jr.
Department of Computer Science
Duke University
Durham, N. C. 27706

HOARE AXIOMS AND THE SEMANTICS OF CONTROL STRUCTURES

1. Introduction. A key trend in program verification has been the use of axioms and rules of inference to specify the meanings of programming language constructs. This approach was first suggested by C.A.R. Hoare in 1969. Although the most complicated control structure in Hoare's original paper was the while statement, there has been considerable success in extending his method to other language features. Axioms have been proposed for the goto statement, functions, recursive procedures with value and reference parameter passing, simple coroutines, and concurrent programs. Recent work by Clarke [CL77] has shown, however, that there are natural programming language control structures which are impossible to adequately describe by means of Hoare axioms. Specifically, Clarke has shown that there are control structures for which it is impossible to obtain axiom systems which are sound and complete in the sense of Cook [CO75]. These constructs include procedures with procedure parameters under standard Algol 60 scope rules, recursive procedures with call by name parameter passing, and coroutines in a language with parameterless recursive procedures.

In this paper we show that Cook's notions of soundness and completeness are natural properties which any adequate Hoare axiom system should possess. We also outline how the incompleteness results are obtained in the case of procedure parameters and suggest ways of modifying Algol 60 scope rules to obtain good axiom systems. Finally, we discuss the significance of incompleteness results for Hoare axiom systems. Although Lipton and Snyder [LS77] have argued that these results have negative implications for the success of program verification, we believe that precisely the opposite is true. We argue that the incompleteness theorems provide additional evidence for the importance of programming languages with simple, clean control structures.

2. Background. The formulas in a Hoare axiom system are triples {P} S {Q}

where S is a statement of the programming language and P and Q are predicates describing the initial and final states of the program S. The logical system in which the predicates P and Q are expressed is called the assertion language (AL) and is an applied version of first order predicate calculus. The triple {P} S {Q} is true iff whenever P holds for the initial program state and S is executed, then either S will fail to terminate or Q will be satisfied by the final program state. We will call such triples partial correctness formulas.

The control structures of the programming language are specified by axioms and rules of inference for the partial correctness formulas. A typical rule of inference is

$$\frac{\{P \wedge b\}\ S\ \{P\}}{\{P\}\ \text{while } b \text{ do } S\ \{P \wedge \sim b\}}$$

Proofs of correctness for programs are constructed by using these axioms together with a proof system T for the assertion language. We write $\vdash_{H,T} \{P\}\ S\ \{Q\}$ if the partial correctness formula $\{P\}\ S\ \{Q\}$ is provable using the Hoare axiom system H and the proof system T for the assertion language AL.

To discuss whether a particular Hoare axiom system H adequately describes the programming language PL, it is necessary to have a definition of truth for partial correctness formulas which is independent of the axiom system H. The definition of truth requires two steps. First, we give an interpretation I for the assertion language AL. The interpretation I specifies the primitive data objects of our programming language; it consists of a set D (the domain of the interpretation) and an assignment of predicates and functions on D to the predicate and function symbols of AL. Typical interpretations might be the integers with the standard functions and predicates of arithmetic, or linear lists with the list processing functions car, cdr, etc.

Second, we provide an interpreter for the statements of the programming language. There are many ways such an interpreter may be specified--in terms of computation sequences (VDL) or as the least fixed point of a continuous functional

(denotational semantics). The net result is a function $M[S](s)=s'$ which associates with each statement S and state s a new state s'. Once the meaning function M has been specified a formal definition may be given for partial correctness.

2.1 Definition: The partial correctness formula $\{P\}\ S\ \{Q\}$ is true with respect to interpretation I ($\models_I \{P\}\ S\ \{Q\}$) iff for all states s and s', if predicate P holds for state s under interpretation I and $M[S](s)=s'$, then Q must hold for s' under I also.

3. Soundness and Completeness. When can we be satisfied that a Hoare axiom system H adequately describes the programming language PL? There are two possible ways a Hoare axiom system may be inadequate. First, some theorem $\{P\}\ S\ \{Q\}$ which can be proven in the axiom system may fail to hold for actual executions of the program S, i.e. there is a terminating computation of S such that the initial state satisfies P but the final state fails to satisfy Q. A way of preventing this source of error is to adopt an operational or denotational semantics for the programming language which is close to the way statements are actually executed. We then show that every theorem, which can be proven using the axiom system, will be true in the model of program execution that we have adopted. In the notation of Section 2 we prove that for all P, Q, S, if $\vdash_{H,T} \{P\}\ S\ \{Q\}$ then $\models_I \{P\}\ S\ \{Q\}$. Logicians call this property soundness or consistency [HL74].

A second source of inadequacy is that the axioms for the programming language may not be sufficiently powerful to handle all combinations of the control structures of the language. The question of when it is safe to stop looking for new axioms is much more difficult to answer than the question of soundness. One solution is to prove a completeness theorem for the Hoare axiom system. We can attempt to prove that every partial correctness formula which is true of the execution model of the programming language is provable in the axiom system. In general it is impossible to prove such completeness theorems; the proof system for

the assertion language may itself fail to be complete. For example, when dealing with the integers for any consistant axiomatizable proof system, there will be predicates which are true of the integers but not provable within the system. Also the assertion language may not be powerful enough to express the invariants of loops. This difficulty occurs if the assertion language is Presburger arithmetic (integer arithmetic without multiplication). Note that both of the above difficulties are faults of the underlying assertion language and not of the Hoare axiom system.

How can we talk about the completeness of a Hoare axiom system independently of its assertion language? Cook [CO75] gives a Hoare axiom system for a subset of Algol including the while statement and nonrecursive procedures. He then proves that if there is a complete proof system for the assertion language (e.g. all true statements of the assertion language) and if the assertion language satisfies a certain natural expressibility condition, then every true partial correctness assertion will be provable. Extensions of Cook's work to other language features are discussed by Gorelick ([GO75], recursive procedures), Owicki ([OW76], concurrent programs), Clarke ([CL77], procedures with procedure parameters under various restrictions on scope of variables), and Cherniavsky ([CH77], loop languages).

To state Cook's expressibility condition we first introduce some additional terminology. WP[S](Q) will denote the weakest precondition for partial correctness corresponding to the statement S and postcondition Q. WP[S](Q) is characterized by (i) $\models \{WP[S](Q)\}\ S\ \{Q\}$ and (ii) $\models \{P\}\ S\ \{Q\}$ implies $\models P \rightarrow WP[S](Q)$.

3.1 Definition: The assertion language AL is expressive with respect to interpretation I iff for all statements S and postconditions Q, there is a formula of AL which expresses WP[S](Q).

It is shown in [CO75] that expressibility insures the existence of invariants

for while loops and recursive procedures. Finally we give a formal definition of Cook's notion of soundness and completeness for Hoare axiom systems.

3.2 Definition: A Hoare axiom system H for a programming language PL is sound and complete iff for all AL, T, and I such that (a) AL is expressive with respect to I and (b) T is a complete proof system for AL with respect to I,

$$\models_I \{P\}\ S\ \{Q\} \iff \vdash_{H,T} \{P\}\ S\ \{Q\}$$

4. Incompleteness Results. In this section we consider the problem of obtaining a sound and complete axiom system for an Algol-like language which allows procedures as parameters of procedure calls. We outline a proof that it is impossible to obtain such a system of axioms even if we disallow calls of the form "call P (...,P,...)"[1]. We also discuss restrictions to the programming language which allow one to obtain a good axiom system.

4.1 Theorem: It is impossible to obtain a system of Hoare-like axioms H which is sound and complete in the sense of Cook for a programming language PL which allows:

(i) procedures as parameters of procedure calls

(ii) recursion

(iii) static scope

(iv) global variables

(v) internal procedures as parameters of procedure calls

All of the features (i) - (v) are found in Algol 60 [NA63] and in Pascal [WI73]. The proof of the theorem uses the following three facts:

Fact (1): If the assertion language AL contains equality and if the interpretation I is finite (i.e. the domain of I is a finite set), then AL is expressive with

---

[1] Calls of this form are necessary if one wants to directly simulate the Lambda calculus by parameter passage.

respect to I.

To see that Fact (1) is true, note that WP[S](Q) may be viewed as the set of states (initial assignments to the global variables of S) which either cause S to run forever or get mapped by S into some state which satisfies Q. Since the assertion language contains equality and the interpretation I is finite, this set of initial assignments may be easily expressed as a formula of AL.

Fact (2): The halting problem is undecidable for the programming language PL with features (i) through (v) for all finite interpretations I.

This fact is the heart of the incompleteness result. Its proof is based on the simulation of a class of turing machines with undecidable halting problem by programs in the language PL operating under a finite interpretation. The Algol 60 execution rule, which states that procedure calls are elaborated in the environment of the procedure's declaration rather than in the environment of the procedure call, allows the simulation program to access values normally buried in the runtime stack without first popping the top of the stack.

Fact (3): Let I be a finite interpretation and let S range over programs in language PL, then the set of true partial correctness formulas {true} S {false} cannot be effectively enumerated.

Fact (3) follows immediately from Fact (2), since the formula {true} S {false} holds iff S fails to halt for any initial state.

We now return to the proof of Theorem 4.1. Choose I to be a finite interpretation and let T be a decision procedure for the truth of formulas in AL relative to I. By Fact (1) AL is expressive relative to I. If there were a sound and complete Hoare axiom system H for PL programs, the true formulas {true} S {false} could be enumerated simply by enumerating the theorems $\vdash_{H,T}$ {true} S {false}. This, however, contradicts Fact (3).

In [CL77] we show that a sound and complete axiom system can be obtained by modifying any one of the five features of the language PL. Thus if we change from static scope to dynamic scope, a complete set of axioms may be obtained for (i) procedures with procedure parameters, (ii) recursion, (iv) global variables, and (v) internal procedures as parameters; or if we disallow internal procedures as parameters, a complete system may be obtained for (i) procedures with procedure parameters, (ii) recursion, (iii) static scope, and (iv) global variables. These results are summarized in Figure 1.

	Language 1	Language 2	Language 3	Language 4	Language 5	Language 6
(1) procedures with procedure parameters	inc.	no procedure parameters	inc.	inc.	inc.	inc.
(2) recursion	inc.	inc.	non-recursive procedures only	inc.	inc.	inc.
(3) global variables	inc.	inc.	inc.	global variables disallowed	inc.	inc.
(4) static scope	inc.	inc.	inc.	inc.	dynamic scope	inc.
(5) internal procedures	inc.	inc.	inc.	inc.	inc.	internal procedures not allowed as parameters
Sound and Complete Hoare-like axiom system?	no	yes	yes	yes	yes	yes

Figure 1 THEOREM SUMMARY

5. Discussion. Perhaps the most important question to be answered is whether a stronger form of expressibility would give completeness for the language PL of Theorem 4.1. The result of Section 4 seems to require that any such notion of expressibility be powerful enough to allow assertions about the status of the runtime stack. Clint [CT75] suggests the use of stack-valued auxiliary variables to prove properties of recursive programs. It seems likely that a notion of expressibility which allowed such variables would give completeness for recursive procedures with procedure parameters. However, the use of such auxiliary variables is counter to the spirit of the high level programming languages. If a proof of a recursive program can involve the use of stack-valued variables, why not simply replace the recursive procedures themselves by stack operations? The purpose of recursion in programming languages is to free the programmer from the details of implementing recursive constructs.

Additional incompleteness results including call by name parameter passing and coroutines with recursive procedures are discussed by Clarke [CL77] and by Lipton and Snyder [LS77]. Lipton and Snyder also argue that there is an inverse relationship between the "power" of a programming language feature and the ease with which programs using the feature can be verified. Although we believe that this observation is correct, we do not believe that restricting a powerful language feature such as procedure parameters or call by name will necessarily hamper the programmer. For example, a typical application of procedure parameters is in numerical integration where the integrand is a parameter of an integration procedure. Here, however, procedures are rarely recursive and the results described in Section 4 may be applied. We believe that an important task of the language designer is to find ways of restricting powerful language features so that neither verifiability nor flexibility of use is sacrificed.

References.

[CH77] Cherniavsky, J. and S. Kamin. A Complete and Consistent Hoare Axiomatics for a Simple Programming Language. Proceedings of the 4th POPL, 1977.

[CL77] Clarke, E. M. Programming Language Constructs for Which it is Impossible to Obtain Good Hoare-like Axiom Systems. Proceedings of the 4th POPL, 1977.

[CK77] Clarke, E. M. Program Invariants as Fixed Points. 18th Annual Symposium on Foundations of Computer Science, 1977.

[CO75] Cook, S. A. Axiomatic and Interpretative Semantics for an Algol Fragment. Technical Report No. 79, Department of Computer Science, University of Toronto, 1975 (to be published in SCICOMP).

[CT73] Clint, M. Program Proving: Coroutines. Acta Informatica 2, 1973 pp. 50-63.

(GO75] Gorelick, G. A. A Complete Axiomatic System for Proving Assertions about Recursive and Non-recursive Programs. Technical Report No. 75. Department of Computer Science, University of Toronto, January 1975.

[HO69] Hoare, C. A. R. An Axiomatic Approach to Computer Programming, CACM 12, 10(October 1969), pp. 322-329.

[HL74] Hoare, C. A. R. and P. E. Lauer. Consistent and Complementary Formal Theories of the Semantics of Programming Languages. Acta Informatica, Vol. 3, pp. 135-154, 1974.

[LS77] Lipton, R. and L. Snyder. Completeness and Incompleteness of Hoare-like Axiom System. Technical Report, Department of Computer Science, Yale University, April, 1977.

[NA63] Naur, P.(ed.) Revised Report on the Algorithmic Language Algol 60. CACM 6, 1(January, 1963), pp. 1-17.

[OW76] Owicki, S. A Consistent and Complete Deductive System for the Verification of Parallel Programs. 8th Annual Symposium on Theory of Computing, 1976.

[WI71] Wirth, N. The Programming Language PASCAL. Acta Informatica 1, 1, 1971, pp. 35-63.